2

AD-A215 109

EFFICIENCY OF THE PRIMAL NETWORK SIMPLEX ALGORITHM FOR THE MINIMUM-COST CIRCULATION PROBLEM

Robert E. Tarjan

CS-TR-187-88

August 1988

N00014-87-K-0467

Efficiency of the Primal Network Simplex Algorithm for the Minimum-Cost Circulation Problem

Robert E. Tarjan [1]

Department of Computer Science
Princeton University
Princeton, New Jersey 08544
and
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

August, 1988

ABSTRACT

We study the number of pivots required by the primal network simplex algorithm to solve the minimum-cost circulation problem. We propose a pivot selection rule with a bound of $n^{(\log n)/2 + O(1)}$ on the number of pivots, for an n-vertex network. This is the first known subexponential bound. The network simplex algorithm with this rule can be implemented to run in $n^{(\log n)/2 + O(1)}$ time. In the special case of planar graphs, we obtain a polynomial bound on the number of pivots and the running time. We also consider the relaxation of the network simplex algorithm in which cost-increasing pivots are allowed as well as cost-decreasing ones. For this algorithm we propose a pivot selection rule with a bound of $O(nm \cdot \min\{\log(nC), m \log n\})$ on the number of pivots, for a network with n vertices, m arcs, and integer arc costs bounded in magnitude by C. The total running time is $O(nm \log n \cdot \min\{(\log nC), m \log n\})$. This bound is competitive with those of previously known algorithms for the minimum-cost circulation problem.

[1] Research partially supported by the National Science Foundation, Grant No. DCR-8605961, and the Office of Naval Research, Contract No. N00014-87-K-0467.

Efficiency of the Primal Network Simplex Algorithm for the Minimum-Cost Circulation Problem

Robert E. Tarjan [1]

Department of Computer Science

Princeton University

Princeton, New Jersey 08544

and

AT&T Bell Laboratories

Murray Hill, New Jersey 07974

August, 1988

1. Introduction

The minimum-cost circulation problem is a central problem in network optimization [15,30,35,40]. Much research has been devoted to finding efficient algorithms for the problem. Classical algorithms for the problem include the cheapest augmenting path algorithm [10,25,26], the out-of- kilter algorithm [17], the cycle-canceling algorithm [29], and the primal network simplex algorithm [12,13,31]. Although some of these methods perform quite well in practice, they all run in time exponential in the problem size in the worst case. Edmonds and Karp [14] were

[1] Research partially supported by the National Science Foundation, Grant No. DCR-8605961, and the Office of Naval Research, Contract No. N00014-87-K-0467.

the first to devise a polynomial-time algorithm for the problem. Their method, which can be viewed [33] as combining the cheapest augmenting path technique with capacity scaling, runs in $O(m \log U\,(m + n \log n))$ time * on a network containing n vertices and m arcs and having integer arc capacities of magnitude at most U.

The work of Edmonds and Karp left open the question of whether the problem has a strongly polynomial algorithm, i.e., an algorithm with a running time bound polynomial in the graph size (m) assuming unit time per arithmetic operation, as well as polynomial in the input size (the number of bits needed to represent the graph and the arc capacities and costs) assuming time per arithmetic operation polynomial in the number of bits in the operands. Tardos [39] was the first to discover a strongly polynomial algorithm. Her work triggered the discovery of other polynomial and strongly polynomial algorithms by Orlin [32,33], Fujishige [16], Bland and Jensen [6], Galil and Tardos [18], Goldberg and Tarjan [20,21], and Ahuja, Goldberg, Orlin, and Tarjan [2]. To date, the best strongly polynomial bound known is Orlin's bound of $O(m \log n\,(m + n \log n))$ [33]. For networks whose arc costs are integers of magnitude at most C, Goldberg and Tarjan's bound of $O(nm \log (n^2/m) \log(nC))$ [20] is best for certain ranges of n,m, and C. For networks whose arc capacities and costs are integral, Ahuja, Goldberg, Orlin, and Tarjan's bound of $O(nm \log\log U \log (nC))$ [2] is best for certain ranges of n,m,U, and C.

These polynomial algorithms depend crucially on the idea of scaling, and their development left open the question of whether any of the classical algorithms can be made polynomial, if pos-

* All time bounds are stated for a unit-cost random-access machine [1].

sible without relying on scaling. Orlin [32] provided a partial answer by proposing a dual network simplex algorithm with a strongly polynomial time bound. His result does use scaling. Goldberg and Tarjan [21] provided another answer by showing that Klein's cycle-canceling algorithm [29] is strongly polynomial if a minimum mean cost criterion is used to select cycles to cancel. They described an implementation of this algorithm with a running time of $O(nm \log n \cdot \min\{\log(nC), m \log n\})$. This algorithm maintains primal feasibility, although the solutions it produces are not necessarily basic.

A remaining open question is whether the *primal* network simplex algorithm has a pivot selection rule that guarantees a polynomial or strongly polynomial bound on the number of pivots. As Orlin [32] observed, this question is motivated both by the practical efficiency of the network simplex algorithm [3,23] and by the polynomial bounds that have been obtained for the average-case behavior of the simplex algorithm for general linear programming [9,38]. Recent work of Goldfarb and Hao [22] adds immediacy to this question. They considered the primal network simplex algorithm for the maximum flow problem and discovered a pivot rule guaranteeing $O(nm)$ pivots and $O(n^2m)$ running time. With the use of dynamic trees [36,37,40], the running time of their algorithm can be improved to $O(nm \log n)$ [19]. This bound is within a factor of $\log n$ or less of all other known bounds for the problem.

In this paper we study the number of pivots required to solve the minimum-cost circulation problem. We propose a pivot rule that has an $n^{(\log n)/2 + O(1)}$ bound * on the number of pivots and the same bound on running time (with a different additive constant in the exponent). Although

* All logarithms in this paper are base two.

this bound is not polynomial, it is the first subexponential bound known. For the special case of planar graphs, we obtain polynomial bounds on the number of pivots and the running time.

We also consider a relaxation of the primal network simplex algorithm in which cost-increasing as well as cost-decreasing pivots are allowed. For this algorithm, we propose a pivot rule with bounds of $O(nm \cdot \min\{\log(nC), m \log n\})$ on the number of pivots and $O(nm \log n \cdot \min\{\log(nC), m \log n\})$ on the total running time. This time bound is the same as that of the Goldberg-Tarjan cycle-canceling algorithm. This is no coincidence; our results are based on theirs. In particular, the notions of ε-optimality and minimum cycle means are central to our work.

This paper consists of five sections in addition to the introduction. Section 2 contains our network terminology. Section 3 presents the primal network simplex algorithm. Section 4 discusses our subexponential bound for general graphs and our polynomial bound for planar graphs. Section 5 discusses our polynomial bound for the relaxed algorithm. Results very similar to those in Section 5 have been obtained independently by Orlin (private communication, 1988) and Goldfarb and Hao (private communication, 1988) using essentially the same methods. Section 6 contains some concluding remarks.

2. Terminology

Our framework for studying the minimum-cost circulation problem is essentially that of Goldberg and Tarjan [20,21]. Let $G = (V,E)$ be a directed graph with vertex set V of size $n \geq 2$ and arc set E of size m. We require G to be *symmetric*, i.e., $(v,w) \in E$ if and only if $(w,v) \in E$.

Symmetry implies that we can think of G as being undirected when it is convenient to do so. We assume without loss of generality that G is connected, i.e., every vertex is reachable from every other vertex. A pair of arcs (v,w), (w,v) is called an *edge* and is represented by the unordered pair $\{v,w\}$. For a vertex v, we denote by $E(v)$ the set $\{w \mid (v,w) \in E\}$. Graph G is a *circulation network* if each arc (v,w) has a real-valued *capacity* $u(v,w)$ and a real-valued *cost* $c(v,w)$. We require the cost function to be *antisymmetric*, i.e., $c(v,w) = -c(w,v)$ for all $(v,w) \in E$. We denote by C the maximum absolute value of an arc cost if all arc costs are integers; if some arc cost is nonintegral, $C = \infty$.

A *circulation* is a real-valued function f on arcs satisfying the following constraints:

$$\forall\ (v,w) \in E,\ f(v,w) \le u(v,w) \text{ (capacity constraints);} \qquad (1)$$

$$\forall\ (v,w) \in E,\ f(v,w) = -f(w,v) \text{ (flow antisymmetry constraints);} \qquad (2)$$

$$\forall\ v \in V,\ \sum_{w \in E(v)} f(v,w) = 0 \text{ (conservation constraints).} \qquad (3)$$

The *cost* of a circulation f is defined by the following formula:

$$cost\ (f) = \sum_{\substack{(v,w) \in E \\ f(v,w) > 0}} c(v,w)\, f(v,w) \qquad (4)$$

The *minimum-cost circulation problem* is that of finding a circulation of minimum cost.

For a circulation f and an arc (v,w), the *residual capacity* of (v,w) is $u_f(v,w) = u(v,w) - f(v,w)$. Arc (v,w) is *residual* if $u_f(v,w) > 0$ and *saturated* if $u_f(v,w) = 0$. We denote by E_f the set of residual arcs and by G_f the *residual graph* (V,E_f). An edge $\{v,w\}$ is *residual* if both (v,w) and (w,v) are residual; otherwise $\{v,w\}$ is *saturated.*

A *cycle* is a sequence $(v_0,v_1), (v_1,v_2), (v_2,v_3),\ldots,(v_{k-1},v_k)$ of arcs, with no arc repeated, such that $v_0 = v_k$. The cycle is *simple* if $v_i \neq v_j$ for $1 \leq i < j \leq k$. The *residual capacity* of a cycle is the minimum of the residual capacities of its arcs. A *residual cycle* is a cycle whose residual capacity is positive. The *cost* of a cycle is the sum of the costs of its arcs. A *negative cycle* is a cycle whose cost is negative. The *mean cost* of a cycle is its cost divided by the number of arcs it contains. The *minimum cycle mean* of a graph with arc costs is the minimum of the mean costs of its cycles, and a *minimum mean cycle* in such a graph is a cycle whose mean cost is minimum.

The following well-known theorem characterizes minimum-cost circulations:

Theorem 2.1 [11]. A circulation is minimum-cost if and only if there are no negative residual cycles.

An alternative characterization of minimum-cost circulations is provided by linear programming duality theory. A *price function p* is any real-valued function on the vertices. Given a price function p, the *reduced cost* of an arc (v,w) is $c_p(v,w) = c(v,w) + p(v) - p(w)$. The cost of any cycle is unaffected by such a transformation of the arc costs.

Theorem 2.2 [15]. A circulation f is minimum-cost if and only if there is a price function p such that

$$\forall\, (v,w) \in E,\ u_f(v,w) > 0 \text{ implies } c_p(v,w) \geq 0 \text{ (optimality constraints).} \qquad (5)$$

We need a notion of approximate optimality introduced by Tardos [39] and independently by Bertsekas [4,5]. This notion is the basis of several minimum-cost circulation algorithms [2,5,20,21,39]. For a real number $\varepsilon \geq 0$, a circulation f is *ε–optimal* if there is a price function p such that

$$\forall\ (v,w) \in E,\ u_f(v,w) > 0 \text{ implies } c_p(v,w) \geq -\varepsilon \quad (\varepsilon\text{–optimality constraints}). \qquad (6)$$

Note that 0-optimality is the same as optimality. As Bertsekas discovered, if all arc costs are integers, then ε-optimality for a sufficiently small ε is enough to give optimality:

Theorem 2.3 [5]: If all arc costs are integers and $\varepsilon < 1/n$, then any ε-optimal circulation is optimal.

The key to our results is the connection between minimum cycle means and ε-optimality, which was discovered by Goldberg and Tarjan. For a circulation f, we denote by $\mu(f)$ the minimum mean cost of a residual cycle, i.e., the minimum cycle mean of the residual graph G_f, and by $\varepsilon(f)$ the minimum ε such that f is ε-optimal.

Theorem 2.4 [20,21]. For any circulation f, $\varepsilon(f) = \max\ \{0, -\mu(f)\}$.

The problem of computing $\varepsilon(f)$ is thus equivalent to the problem of computing the minimum cycle mean of G_f. There are three known efficient algorithms for computing minimum cycle means: Karp's $O(nm)$-time algorithm [27], Karp and Orlin's $O(nm \log n)$-time algorithm [28], and Orlin and Ahuja's $O(n^{1/2}m \log(nC))$-time algorithm [34]; the last of these requires integer arc costs. Given an ε such that f is ε-optimal, a price function p satisfying the ε-

optimality constraints can be found in $O(nm)$ time using the Bellman-Ford shortest path algorithm, as discussed by Goldberg and Tarjan [20].

We need one more concept related to the network simplex algorithm. A circulation f is *basic* if the set of residual edges forms a forest (a set of vertex-disjoint trees).

3. Cycle-Canceling and the Primal Network Simplex Algorithm

Theorem 2.1 suggests the following *cycle-canceling algorithm*, discovered by Klein [29], for finding a minimum-cost circulation. Begin with any circulation. (If there exists a circulation, one can be found using any maximum flow algorithm.) Repeat the following step until there are no negative residual cycles: Find a negative residual cycle Γ and *cancel* it by increasing the flow on each arc of Γ by an amount equal to the residual capacity of Γ. Although this algorithm can run for an exponential number of steps if a bad choice of cycles to cancel is made, Goldberg and Tarjan [21] showed that if a minimum mean cost residual cycle is canceled at each step, the number of steps is polynomial, specifically $O(nm \cdot \min\{\log(nC), m\log n\})$. A related rule for choosing cycles to cancel can be implemented to that the total running time of the algorithm is $O(nm \log n \cdot \min\{\log(nC), m \log n\})$.

The cycle-canceling algorithm maintains primal feasibility in the linear programming sense, i.e., it maintains a circulation at each step, but it does not maintain a basic solution, i.e., a basic circulation. The *primal network simplex algorithm*, henceforth just called the *network simplex algorithm*, is a variant of the cycle-canceling algorithm that does maintain a basic solution. The simplex algorithm maintains a basic circulation f and a spanning tree T whose edges are edges of

G, such that T contains all the residual edges. An arc (v,w) is said to be a *tree arc* if the corresponding edge $\{v,w\}$ is in T and a *nontree arc* otherwise. A nontree arc (v,w) defines a *basic cycle* $\Gamma(v,w)$ consisting of the arc (v,w) and the simple path of tree arcs from w to v. Arc (v,w) is *pivotable* if $\Gamma(v,w)$ is negative. The simplex algorithm consists of repeating the following *pivot step* until no nontree arc is pivotable: Select a pivotable nontree arc (v,w). Cancel $\Gamma(v,w)$ by increasing the flow on every arc of $\Gamma(v,w)$ by an amount equal to the residual capacity of $\Gamma(v,w)$. (This amount may be zero; in this case the pivot is called *degenerate*.) Add edge $\{v,w\}$ to T and delete some edge $\{x,y\}$ such that arc (x,y) is on Γ and (x,y) is saturated. Arc (v,w) is called the *entering arc* of the pivot and arc (x,y) is called the *leaving arc*; it is possible for the two to be equal. The pivot is said to be a pivot *on* (v,w).

The simplex algorithm differs from the cycle-canceling algorithm in that, although both algorithms work by canceling cycles, the simplex algorithm cancels only *basic* negative cycles, whereas the cycle-canceling algorithm cancels only *residual* negative cycles. A degenerate pivot, i.e., one canceling a nonresidual basic cycle, does not change the circulation f, only the spanning tree T. The *relaxed simplex algorithm*, which we study in Section 5, is the relaxation of the simplex algorithm in which a pivot is allowed on any nontree residual arc (v,w), whether or not $\Gamma(v,w)$ is negative.

Standard implementations of the simplex algorithm maintain a price function p such that every tree arc has a reduced cost of zero. Such a price function can be defined by rooting the tree T at an arbitrary vertex r and defining $p(v)$ for any vertex v to be the total cost of the tree arcs along the simple path in the tree from r to v. With such a price function, a nontree residual arc

(v,w) is pivotable if and only if it has negative reduced cost. Furthermore, on termination of the algorithm the circulation f and the price function p satisfy the optimality constraints.

The simplex algorithm requires an initial basic circulation to get started. Such a basic circulation can be found by using any maximum flow algorithm to find some circulation and then repeatedly finding an undirected simple cycle of residual edges and increasing (or decreasing) the flow around the cycle so that one of its edges becomes saturated. All such undirected residual cycles can be canceled in $O(m \log n)$ time by using a slight variant of the algorithm of Sleator and Tarjan [36] for making a flow acyclic. Having found a basic circulation, a suitable initial spanning tree can be found by choosing any spanning tree containing all the residual edges. Such a tree can be constructed in $O(m)$ time using graph search.

Remark. The idea of canceling undirected residual cycles can also be used to convert any minimum-cost circulation into a basic minimum-cost circulation. Given a minimum-cost circulation f, we first compute a price function p with respect to which f satisfies the optimality constraints. This takes $O(nm)$ time as noted in Section 2. Any residual edge $\{v,w\}$ is such that both (v,w) and (w,v) have reduced cost zero. We then cancel undirected residual cycles as described above. Canceling any such cycle does not change the cost of the circulation, since any such cycle has zero reduced cost. All the cycle-canceling takes $O(m \log n)$ time. □

The possibility of degeneracy, as indicated by the presence of one or more saturated edges in the spanning tree T, requires us to adjust some of the definitions of Section 2. We say that an arc (v,w) is *pseudo-residual* if (v,w) is residual or $\{v,w\}$ is in T. We denote by E_f^* the set of pseudo-residual arcs. A *pseudo-residual cycle* is a simple cycle consisting of pseudo-residual arcs.

4. A Subexponential-Time Pivot Rule

Our pivot rule for the network simplex algorithm is based on the cycle-canceling algorithm of Goldberg and Tarjan. Consider the refinement of the simplex algorithm that consists of repeating the following step until there is no negative pseudo-residual cycle: Choose a minimum mean cost pseudo-residual cycle Γ and *block* it by repeatedly doing pivots, each of which is on an arc of Γ, until some arc of Γ is not pseudo-residual, i.e., some arc of Γ is a saturated nontree arc.

We postpone a discussion of how to choose pivots to block a cycle Γ. First, we establish a polynomial bound on the number of cycle blockings. Our bound is $O(nm \cdot \min\{\log(nC), m \log n\})$, which is the same as the bound on cycle cancellations for the Goldberg-Tarjan algorithm. The derivation of our bound is analogous to the derivation of the Goldberg-Tarjan bound. In the discussion to follow, f is a basic circulation and T is a spanning tree containing all the residual edges. The analysis requires the use of a price function p that is *not* the same as the price function used in standard implementations of the network simplex algorithm. (The latter price function makes the reduced cost of every tree arc zero.) This distinction needs to be remembered.

For a real number $\varepsilon \geq 0$, circulation f is *strongly* ε- *optimal* if there is some price function p for which $c_p(v,w) \geq -\varepsilon$ for every pseudo-residual arc (v,w). Strong ε-optimality implies ε-optimality. We denote by $\varepsilon^*(f)$ the minimum ε for which f is strongly ε-optimal. We denote by

$\mu^*(f)$ the minimum mean cost of a pseudo-residual cycle. Theorem 2.4 translates into the following result:

Theorem 4.1. $\varepsilon^*(f) = \max\{0, -\mu^*(f)\}$.

Proof. The value $\varepsilon(f)$ is a function only of the set of residual arcs and their costs; $\varepsilon^*(f)$ is the same function of the set of pseudo-residual arcs and their costs. The analogous statement is true of $\mu(f)$ and $\mu^*(f)$. The theorem is thus immediate from Theorem 2.4. □

Let $\varepsilon = \varepsilon^*(f)$, and let p be a price function with respect to which f is strongly ε-optimal. Holding ε and p fixed, we study the effect of cycle blockings on $\varepsilon^*(f)$.

Lemma 4.2. Blocking a cycle cannot increase $\varepsilon^*(f)$.

Proof. (See the proof of Lemma 3.5 of [21].) Let Γ be the cycle blocked. Every arc of Γ has reduced cost exactly $-\varepsilon$. Every new pseudo-residual arc (v,w) created by a pivot step during the blocking of Γ has $-\varepsilon \le c_p(v,w) \le \varepsilon$. This follows by cost antisymmetry and the fact that, just before the blocking, every tree arc (x,y) satisfies $-\varepsilon \le c_p(x,y) \le \varepsilon$ by strong ε-optimality. Thus, after the blocking of Γ, the new circulation f is still strongly ε-optimal with respect to p and the new T. Therefore $\varepsilon^*(f) \le \varepsilon$ after the blocking. □

Lemma 4.3. A sequence of $m/2-2n+2 \le m$ cycle blockings reduces $\varepsilon^*(f)$ to at most $(1-1/n)\,\varepsilon$.

Proof. (See the proof of Lemma 3.6 of [21] .) Consider a sequence of $m/2-2n+2$ cycle blockings. Let Γ be a minimum mean cost pseudo-residual cycle just after one of these blockings.

(Cycle Γ might or might not be the next cycle to be blocked.) Suppose Γ contains at least one arc of nonnegative reduced cost. Let $l \le n$ be the number of arcs on Γ. By strong ε-optimality, which is maintained by Lemma 4.2, every arc of Γ has reduced cost at least $-\varepsilon$. Thus the mean cost of Γ is at least $-((l-1)/l)\,\varepsilon$ $-(1-1/n)\varepsilon$. This implies $\varepsilon^*(f) \le (1-1/n)\varepsilon$.

It follows that if any of the $m/2-2n+2$ cycles blocked contains an arc of nonnegative reduced cost, then the lemma is true. (Once $\varepsilon^*(f) \le (1-1/n)\varepsilon$, this inequality remains true by Lemma 4.2.) Suppose that each of the $m/2-2n+2$ cycles blocked contains only arcs of negative reduced cost. Then each pivot during the blockings is on an arc of negative reduced cost. Furthermore, each cycle blocking produces a saturated nontree arc (v,w) of negative reduced cost. No later pivot can be on either (v,w) or (w,v). Since the tree T always contains exactly $n-1$ edges, there are always exactly $2n-2$ tree arcs. Thus, after all $m/2-2n+2$ cycle blockings, every nontree residual arc must have nonnegative reduced cost. This means that any minimum mean cost pseudo-residual cycle contains an arc of nonnegative reduced cost. By the argument in the preceding paragraph, $\varepsilon^*(f) \le (1-1/n)\varepsilon$. □

Theorem 4.4. If all arc costs are integers, then the cycle blocking algorithm terminates after $O(nm\log(nC))$ cycle blockings.

Proof. (See the proof of Theorem 3.7 of [21].) Let f be the circulation maintained by the algorithm. Initially $\varepsilon^*(f) \le C$. If $\varepsilon^*(f) < 1/n$, then $\varepsilon(f) = 0$ by Theorem 2.3, since strong ε-optimality implies ε-optimality. By Lemma 4.3, each group of m cycle blockings reduces $\varepsilon^*(f)$ by a factor of at least $1-1/n$. It follows that the algorithm terminates after $O(nm\log(nC))$ cycle blockings.

(Each group of nm cycle blockings reduces $\varepsilon^*(f)$ by at least a constant factor.) □

To obtain a strongly polynomial bound on the number of cycle blockings, we need two more definitions. We say that an arc (v,w) is *ε-fixed* if $f(v,w)$ is the same for every ε-optimal circulation f. The following theorem of Goldberg and Tarjan generalizes a result of Tardos [39].

Theorem 4.5 [20,21]. Let $\varepsilon > 0$. Suppose a circulation is ε-optimal with respect to a price function p. Suppose further that, for some arc (v,w), $| c_p(v,w) | \geq 2n\varepsilon$. Then (v,w) is ε-fixed.

We say an arc (v,w) is *dead* if it is a saturated nontree arc and its flow does not change during the remainder of the algorithm; we say (v,w) is *live* otherwise. Every arc on a cycle to be blocked is live since each such arc is either residual or a tree arc. Thus every pivot is on a live arc.

Theorem 4.6. For arbitrary real-valued arc costs, the cycle blocking algorithm terminates after $O(nm^2 \log n)$ cycle blockings.

Proof. (See the proof of Theorem 3.9 in [21].) Divide the cycle blockings into groups of $k = cmn \log n$, where c is a suitably large positive constant, to be chosen later. We shall show that each group of k cycle blockings converts at least one arc from live to dead. The theorem follows.

Consider a group of k cycle blockings. Let f and f' be the circulations before and after the k cycle blockings, and let $\varepsilon = \varepsilon^*(f)$, $\varepsilon' = \varepsilon^*(f')$. By Lemma 4.3, $\varepsilon' \leq \varepsilon/(2n)$ for c a sufficiently large constant. Let Γ be the first cycle blocked. The mean cost of Γ is $-\varepsilon$. Let p' be a price function with respect to which f' is strongly ε'-optimal. Some arc of Γ, say (v,w), has a reduced cost (with respect to p') of at most $-\varepsilon \le -2n\varepsilon'$. By Theorem 4.5, (v,w) is ε-fixed after all the cycle blockings. Since the circulation remains strongly ε-optimal by Lemma 4.2, the flow through (v,w) cannot subsequently change. Since f' is strongly ε'-optimal with respect to p', and since the reduced cost of (v,w) with respect to p' is strictly less than $-\varepsilon'$, (v,w) is not pseudo-residual, i.e., it is a saturated nontree arc. Thus (v,w) is dead after the cycle blockings. This proves the claim. □

Having shown that the number of cycle blockings is polynomial, we must still devise a way to perform cycle blockings in a "small" number of pivots. Let Γ be an arbitrary negative pseudo-residual cycle. The following method will block Γ in at most $n^{(\log n)/2}$ pivots.

We need to work with condensations of the graph G. If (v,w) is an arc of G, the *condensed graph* $G(v,w)$ is formed by identifying vertices v and w. The operation of forming $G(v,w)$ is called *condensing* (v,w). Note that $G(v,w)$ will contain loops (arcs of the form (x,x)) and may contain multiple arcs (two or more arcs of the form (x,y)). We allow such loops and multiple arcs to remain; we call $G(v,w)$ a *multigraph*. If S is a subset of the arcs of G, we form the *condensed graph* $G(S)$ by condensing each of the arcs in S in turn. Up to renaming of vertices, the order of these condensations is irrelevant.

Let T be the current spanning tree. Let p be a price function such that every tree arc has zero cost. In what follows, "reduced cost" means with respect to p; the reduced cost of an arc in a condensed graph is defined to be the reduced cost of the corresponding arc in the original graph.

We begin the blocking of cycle Γ by defining a sequence of triples (G_1,T_1,Γ_1), (G_2,T_2,Γ_2),..., (G_k,T_k,Γ_k). For $1 \le i \le k$, G_i is a condensation of G formed by condensing some of the arcs of T; T_i is the spanning tree of G_i whose arcs correspond to those of T; Γ_i is a negative reduced-cost simple cycle in G_i whose arcs correspond to those of Γ. The first triple is $(G_1,T_1,\Gamma_1) = (G,T,\Gamma)$. For $i > 1$, the i^{th} triple is defined if the $i-1^{st}$ triple is defined and $|\Gamma_{i-1}-T_{i-1}| \ge 2$; here Γ_{i-1} and T_{i-1} are interpreted as sets of arcs, and absolute value signs denote set size. If $|\Gamma_{i-1}-T_{i-1}| \ge 2$, $G_i = G_{i-1}\ (T_{i-1} - \Gamma_{i-1})$. The tree T_i is the spanning tree of G_i consisting of the nonloop arcs that correspond to arcs of T_{i-1}. The cycle Γ_i is chosen to be any negative reduced-cost simple cycle in G_i, all of whose arcs correspond to arcs of Γ_{i-1}.

Lemma 4.7. For $2 \le i \le k$, a suitable cycle Γ_i can be chosen.

Proof. By induction on i. There is a cycle Γ_i' in G_i, not necessary simple, whose arcs correspond to those of Γ_{i-1}. Cycles Γ_{i-1} and Γ_i' have the same reduced cost. By the induction hypothesis, Γ_{i-1} has negative reduced cost; thus so does Γ_i'. The arcs of Γ_i' can be partitioned into simple cycles, at least one of which must have negative reduced cost and hence can be chosen as Γ_i.

Since cycle Γ_i for $1 \le i \le k$ is simple, it contains a loop only if the loop is the entire cycle. Thus Γ_i contains no zero reduced-cost loops, which means that any arc $(v,w) \in \Gamma_i \cap T_i$ has $v \ne w$.

Suppose we perform a pivot on the arc (v,w) in Γ corresponding to the unique arc in $\Gamma_k - T_k$. Since Γ_k has negative reduced cost, so does (v,w), and we can indeed pivot on (v,w). Let (x,y) be the leaving arc of the pivot. Let j be maximum such that x and y are not condensed in G_j. Then the pivot in G that replaces $\{v,w\}$ by $\{x,y\}$ in T corresponds to a pivot in each G_i for $1 \le i \le j$ that forms a new tree T_i' by replacing the edge corresponding to $\{v,w\}$ by the edge corresponding to $\{x,y\}$. This observation leads to the following pivot rule for blocking Γ. Construct a sequence of triples (G_1,T_1,Γ_1), (G_2,T_2,Γ_2),...,(G_k,T_k,Γ_k) as specified above. Then repeat the following step until some arc of Γ becomes a saturated nontree arc:

Pivot and Restore Triples. Pivot on the arc (v,w) in Γ corresponding to the unique arc in $\Gamma_k - T_k$. Let (x,y) be the leaving arc of the pivot. Let j be maximum such that x and y are not condensed in G_j. Perform the corresponding pivot in G_i for $1 \le i \le j$ to form T_i' from T_i. Restore the sequence of triples by beginning with (G_1,T_1',Γ_1),... (G_j,T_j',Γ_j) and extending the sequence using the inductive definition above. (The new number of triples k may differ from the old number.)

In order to measure the progress of this algorithm, we define $\alpha_i = |\Gamma_i - T_i|$ for $1 \le i \le k$. We define the *signature* of a sequence of k triples to be the sequence $\alpha_1,\alpha_2,...,\alpha_k$. We order signatures lexicographically, i.e., $\alpha_1,\alpha_2,...,\alpha_k < \beta_1,\beta_2,...,\beta_l$ if either there is some $j \le \min\{k,l\}$ such that $\alpha_i = \beta_i$ for $1 \le i < j$ and $\alpha_j < \beta_j$, or $k < l$ and $\alpha_i = \beta_i$ for $1 \le i \le k$.

Lemma 4.8. Each pivot step decreases the signature or terminates the cycle blocking.

Proof. Consider a pivot on an arc (v,w). Let $\alpha_1,\alpha_2,...,\alpha_k$ and $\beta_1,\beta_2,...,\beta_l$ be the signatures before and after the pivot, respectively. Let (x,y) be the leaving arc, and let j be maximum such that x and y are not condensed in G_j. If $j < k$, then the arc in G_j corresponding to (x,y) is in $T_j - \Gamma_j$, since it is one of the arcs condensed to form G_{j+1}. Furthermore, for $1 \le i < j$, the arc in

G_i corresponding to (x,y) is in $T_i \cap \Gamma_i$. It follows that $\alpha_i = \beta_i$ for $1 \le i < j$ and $\alpha_j < \beta_j$.

Suppose on the other hand, that $j = k$. Then the basic cycle in G_k on which the pivot occurs is Γ_k, and arc (x,y) must lie on Γ. The pivot makes (x,y) a saturated nontree arc, which terminates the blocking. □

Lemma 4.9. For $1 \le i \le k$, $\alpha_i = |\Gamma_i - T_i| \le n/2^{i-1}$.

Proof. For $1 \le i \le k$, let τ_i be the number of vertices v in G_i such that v is incident to some arc of $T_i - \Gamma_i$. We claim that $\alpha_i \le \tau_i$ or $\alpha_i = 1$. To verify the claim, suppose that $\alpha_i \ge 2$. The cycle Γ_i consists of arcs not in T_i alternating with paths of arcs in T_i. (Some of these paths may consist of single vertices and no arcs.) Some vertex along each path in $\Gamma_i \cap T_i$ must be incident to an arc of $T_i - \Gamma_i$, since there are $\alpha_i \ge 2$ such paths, which are connected in T_i. Hence $\alpha_i \le \tau_i$.

Now we show that $\tau_i \le \tau_{i-1}/2$ for $2 \le i \le k$. This inequality implies $\tau_i \le n/2^{i-1}$ for $1 \le i \le k$ by induction on i, since $\tau_1 \le n$. This in turn gives the lemma. To show that $\tau_i \le \tau_{i-1}/2$, we observe that every vertex v in G_{i-1} incident to an arc in $T_{i-1} - \Gamma_{i-1}$ is condensed into another such vertex in forming G_i, namely the other vertex incident to the arc. Furthermore if v is a vertex in G_i incident to an arc in $T_i - \Gamma_i$, then the corresponding vertex in G_{i-1} is incident to the corresponding arc, and this arc is in $T_{i-1} - \Gamma_{i-1}$ because every arc in Γ_i corresponds to an arc in Γ_{i-1}. It follows that each vertex counted in τ_i is formed by condensing two or more vertices counted in τ_{i-1}, which implies that $\tau_i \le \tau_{i-1}/2$. □

Theorem 4.10. The cycle-blocking algorithm blocks Γ in at most $n^{(\log n)/2}$ pivots.

Proof. By Lemma 4.8, each pivot either terminates the algorithm or decreases the signature. Lemma 4.9 implies that there are at most $\prod_{i=1}^{\lceil \log n \rceil} n/2^{i-1} \le n^{(\log n)/2}$ distinct possible signatures. The theorem follows. □

Combining Theorems 4.4, 4.6, and 4.10, we obtain the main result of this section:

Theorem 4.11. The network simplex algorithm with pivot selection based on minimum mean cost cycle blocking terminates in $O(n^{(\log n)/2+1} m \cdot \min\{\log(nC), m \log n\})$ pivots and $O(n^{(\log n)/2+1} m^2 \cdot \min\{\log(nC), m \log n\})$ running time.

Proof. The bound on pivots is immediate. It is not hard to implement the algorithm so that the amortized time [*] per pivot is $O(m)$; we leave this as an exercise. □

We conjecture that the amortized time per pivot can be reduced to $O(n)$ or less, but we shall not pursue this issue here, since the bound on pivots is so large. Instead, we consider the case of planar graphs and show that for such graphs the number of pivots can be made polynomial.

Suppose that G is planar. Planarity implies that $m = O(n)$ [7]. We shall show that a judicious choice of Γ_2 and Γ_3 in the cycle-blocking algorithm guarantees that every signature contains at most three terms, i.e., one of α_1, α_2, and α_3 is 1. This implies that n^2 pivots suffice to block a cycle, since every term in a signature except the last is between 2 and n, inclusive, and the last is 1.

[*] By "amortized time" we mean the time per operation averaged over a worst case sequence of operations. See [41].

Form G_1' from $G_1 = G$ by deleting all arcs except those in $T_1 \cup \Gamma_1$. Regard G_1' as an undirected graph and embed it in the plane. Assume without loss of generality that G_1' is embedded so that Γ_1 is directed clockwise around its interior. Consider the boundaries of the faces inside Γ_1, oriented clockwise around their interiors. These boundaries are simple cycles that partition the arcs of Γ_1 and the tree arcs inside Γ_1. The sum of the costs of these cycles is exactly the cost of Γ_1; thus at least one such cycle must have negative cost. Let Γ_1' be such a negative cycle. Choose Γ_2 to be any negative simple cycle in G_2 whose arcs correspond to those of Γ_1'.

If $|\Gamma_2 - T_2| \geq 2$, proceed as follows to choose Γ_3. Form G_2' from G_1' by condensing the arcs of T_1 lying inside Γ_1 and deleting all arcs except those corresponding to noncondensed arcs of $\Gamma_1' \cup T_1$. Let T_2' be the cycle in G_2' corresponding to Γ_1', and let G_2' be the spanning tree of G_2' corresponding to the noncondensed arcs of T_1. Condensing the planar embedding of G_1' gives a planar embedding of G_2' (regarded as an undirected graph) such that Γ_2' is the boundary of a face, with Γ_2' oriented clockwise around its interior. Consider the boundaries of the other faces of G_2', oriented counterclockwise around their interiors. These boundaries are simple cycles that partition the arcs of $\Gamma_2' \cup T_2'$. Because the embedding is planar and T_2' is a spanning tree, each such cycle contains exactly one arc of $\Gamma_2' - T_2'$. Let Γ_3' be such a cycle that contains a negative-cost arc corresponding to an arc of Γ_2. Choose Γ_3 to be the simple cycle in G_3 corresponding to Γ_3'. Then $|\Gamma_3 - T_3| = 1$.

Theorem 4.12. If G is a planar graph, and Γ_2 and Γ_3 are chosen in the cycle-blocking algorithm as described above, then the number of pivots per cycle blocking is at most n^2, the total number of pivots needed to find a minimum-cost circulation is $O(n^4 \min \{ \log(nC), n\log n\})$, and the time needed to find a minimum-cost circulation is $O(n^5 \cdot \min \{\log(nC), n\log n\})$.

Proof. The discussion above and Lemma 4.8 imply that the number of pivots per cycle blocking is at most n^2. The bound on total pivots follows from Theorems 4.4 and 4.6 and $m = O(n)$. The time per pivot is $O(n)$ if a linear-time planarity algorithm [8,24] is used to embed G. (Actually, it suffices to embed G once and modify the embedding as needed to choose entering arcs for pivot steps.) □

5. A Polynomial-Time Relaxed Network Simplex Algorithm

The results in Section 4 relate the efficiency of the network simplex algorithm to that of cycle blocking. If we relax the network simplex algorithm to allow cost-increasing pivots in addition to cost-decreasing ones, then cycle blocking becomes much easier. The following argument shows that at most n pivots suffice to block a cycle. Let Γ be the cycle to be blocked, and let v be an arbitrary vertex on Γ. The rule for pivoting is to pivot on the first nontree arc after v on Γ. If this rule is used, then after k pivots either Γ is blocked or the first k arcs after v on Γ are tree arcs. Thus, after n or fewer pivots, Γ must be blocked. This gives the following result:

Theorem 5.1. With the pivot rule for cycle blocking proposed above, the relaxed network simplex algorithm takes $O(n^2 m \cdot \min \{\log(nC), m\log n\})$ pivots and $O(n^2 m^2 \cdot \min \{\log(nC), m\log n\})$ total time.

Proof. The bound on pivots follows from Theorems 4.4 and 4.6 and the discussion above. The time for blocking a cycle is $O(nm)$ to find a suitable (minimum mean cost) cycle to block plus $O(m)$ for each of the n pivots, for a total of $O(nm)$. The total time bound follows from Theorems 4.4 and 4.6. □

We shall reduce the bound on pivots in Theorem 5.1 by a factor of n and the bound on time by a factor of $nm/\log n$. To do this we first relax the rule for choosing cycles to block. We follow the approach Goldberg and Tarjan [21] used to develop a more efficient version of the cycle-canceling algorithm.

Recall that an arc (v,w) is *pseudo-residual* if it is residual or a tree arc, and that a basic circulation f is *strongly* ε*-optimal* if there is a price function p for which $c_p(v,w) \geq -\varepsilon$ for every pseudo-residual arc (v,w). The new algorithm explicitly maintains such a price function p. (In Section 4, such a price function was used only to analyze the algorithm, not to define it.) For a basic circulation f and a price function p, we denote by $\varepsilon^*(f,p)$ the minimum $\varepsilon > 0$ such that $c_p(v,w) \geq -\varepsilon$ for every pseudo-residual arc (v,w). We call a pseudo-residual arc (v,w) *admissible* if $c_p(v,w) < 0$. The idea of the algorithm is to repeatedly perform pivot steps on admissible arcs until there is no cycle of admissible arcs, and then modify p to reduce $\varepsilon^*(f,p)$. To be precise, given an initial basic circulation f and a spanning tree T containing the residual edges, the algorithm first computes a price function p such that $\varepsilon^*(f) = \varepsilon^*(f,p)$. Then the algorithm repeats the following two steps until f is optimal:

Step 1 (block admissible cycles). Repeatedly perform pivots on admissible arcs until there is no cycle of admissible arcs.

Step 2 (tighten prices). Modify p so that $\varepsilon^*(f,p)$ decreases to at most $(1-1/n)$ times its former value.

A proof like that of Lemma 4.2 shows that Step 1 cannot increase $\varepsilon^*(f,p)$. We postpone a discussion of the implementation of Step 1 in favor of Step 2.

Step 2 can be implemented either using a minimum cycle mean calculation, which takes $O(nm)$ time (see Section 2), or with the following simple two-step, $O(m)$-time computation.

Step 2a (compute levels). For each vertex v, compute a *level* $L(v)$, defined recursively by $L(v) = 0$ if v has no incoming admissible arcs, $L(v) = 1 + \max \{L(u) \mid (u,v) \text{ is an admissible arc}\}$ otherwise.

Step 2b (compute new prices). Let $\varepsilon = \varepsilon^*(f,p)$. For each vertex v, replace $p(v)$ by $p(v) - (\varepsilon/n)\,L(v)$.

Lemma 5.2. Either implementation of Step 2 is correct.

Proof. Lemma 4.2 of [21] states that Steps 2a and 2b correctly implement Step 2. The implementation using a minimum cycle mean calculation reduces $\varepsilon^*(f,p)$ to $\varepsilon^*(f)$, which is the minimum possible value of $\varepsilon^*(f,p)$. Since Steps 2a and 2b will reduce $\varepsilon^*(f,p)$ to at most $(1-1/n)$ times its former value, so must the minimum cycle mean calculation. □

Suppose we implement Step 2 by using Steps 2a and 2b, except that every n^{th} iteration of Step 2 we use a minimum cycle mean computation instead. Then the amortized time per iteration of Step 2 is $O(m)$. The number of iterations of Steps 1 and 2 is $O(n \log(nC))$. This is because $\varepsilon^*(f,p) \le C$ initially, the algorithm terminates when $\varepsilon^*(f,p) < 1/n$, and every n iterations decrease $\varepsilon^*(f,p)$ by at least α constant factor. Furthermore, a proof like that of Lemma 4.6 shows that the number of iterations of Steps 1 and 2 is $O(nm \log n)$. Thus we obtain the following result:

Theorem 5.3. The relaxed network simplex algorithm terminates in $O(n \cdot \min\{\log(nC), m \log n\})$ iterations of Steps 1 and 2. The amortized time per iteration of Step 2 is $O(m)$.

It remains to implement Step 1. We shall describe a way to carry out Step 1 in $O(m)$ pivots. Then we shall describe how to extend the dynamic tree data structure of Sleator and Tarjan [36,37,40] so that it can be used to perform each pivot step in $O(\log n)$ amortized time and all of Step 1 in $O(m \log n)$ time.

Our implementation of Step 1 maintains a partition of the vertices into two classes: *marked* and *unmarked*. Initially all vertices are unmarked. A vertex v becomes marked when the algorithm discovers that there is no path of one or more admissible arcs from v to an unmarked vertex. The algorithm maintains a root vertex r for the spanning tree T, such that the following invariant holds:

(*) If (v,w) is an admissible tree arc such that w is unmarked, then either w is the parent of v in

T (i.e., w lies on the tree path from v to r), or (v,w) has always been an admissible tree arc with v the parent of w. (In particular, if the latter case holds then (v,w) was not added to T by a pivot.)

Each pivot is on an arc of the form (r,v). The algorithm consists of the following steps:

Step 1a (initialize). Unmark all vertices. Select any vertex r and root T at r.

Step 1b (pivot). If there is no admissible arc of the form (r,v) with v unmarked, mark r and go to Step 1c. Otherwise, let (r,v) be such an arc. If (r,v) is a tree arc, root T at v, replace r by v, and repeat Step 1b. If, on the other hand, (r,v) is a nontree arc, pivot on (r,v). Let (x,y) be the leaving arc of the pivot. Root the tree at x, replace r by x, and go to Step 1c.

Step 1c (reroot). If every vertex is marked, stop. Otherwise, let v be any unmarked vertex. Let x be the unmarked vertex closest to r along the tree path from v to r. If $x \neq r$, root the tree at x and replace r by x. Go to Step 1b.

Lemma 5.4 Steps 1a-1c correctly implement Step 1 and maintain invariant (*).

Proof. Pivoting on an admissible arc cannot create any new admissible arcs. Consider Step 1b. It there is no admissible arc (r,v), then marking r preserves the invariant that there is no path of admissible arcs from a marked vertex to an unmarked vertex. If (r,v) is an admissible tree arc, then rerooting the tree at v obviously preserves (*). If (r,v) is an admissible nontree arc, rerooting the tree after a pivot on (r,v) also preserves (*), since the entering arc (r,v) has v the parent of r

after the rerooting (unless (r,v) is the leaving arc), and the rerooting leaves invariant the parent and the child vertices of all other tree arcs.

Rerooting the tree as in Step (1c) also preserves (*), since each tree arc (y,z) along the tree path from v to r has z marked, and the reversal (z,y) of such an arc has y the parent of z after the rerooting. □

Lemma 5.5. The number of iterations of Steps 1b and 1c is $O(m)$.

Proof. We need only count iterations of Step 1b, since every iteration of 1c follows an iteration of 1b. Each iteration of 1b either marks a vertex, which can happen at most n times, or creates a new admissible tree arc of the form (x,y) with y unmarked and y the parent of x, or does a pivot. Let (x,y) be the leaving arc of a pivot. If (x,y) is not admissible or y is marked, (x,y) will never later be the entering arc of a pivot. If (x,y) is admissible and y is unmarked, then invariant (*) implies that (x,y) is saturated after the pivot and hence inadmissible. In this case also it cannot later be the entering arc of a pivot. Thus there can only be one pivot per arc, for a total of at most m. Once an admissible arc (x,y) becomes a tree arc with y the parent of x and y unmarked, it remains in this state until it becomes the leaving arc of a pivot. It follows that the total number of iterations of Step 1b is at most $n + 3m$. □

It is straightforward to implement Steps 1a-1c so that the total time for Step 1 is $O(nm)$. To do this we maintain the tree T using a set of parent pointers, one for each node. Then each iteration of Step 1b or 1c takes $O(n)$ time.

We can reduce the amortized time for pivoting and rerooting the tree by using a dynamic tree data structure. The data structure represents a collection of vertex-disjoint rooted trees, each edge $\{v,w\}$ of which has two associated real values, $g(v,w)$ and $g(w,v)$, and each vertex of which has a mark bit. We denote by *parent*(v) the parent of vertex v in its tree; if v is a tree root, *parent*(v) = null. We adopt the convention that every tree vertex is both an ancestor and a descendant of itself. The data structure supports the following ten operations:

make-tree(v): Make vertex v into a one-vertex tree, with v unmarked. Vertex v must be in no other tree.

find-parent(v): Return the parent of vertex v, or null if v is a tree root.

find-value(v): Compute and return $g(v,$*parent*$(v))$; if v is a tree root, return infinity.

find-min(v): Find and return an ancestor w of vertex v such that $g(w,$ *parent*$(w))$ is minimum; if v is a tree root, return v.

find-vertex(v): Find and return the unmarked ancestor of vertex v, if any, nearest the root.

change-mark(v,b): If bit b is 1, make vertex v marked; if b is 0, make v unmarked.

change-value(v, δ): Add real number δ to $g(w$ *parent*$(w))$ and subtract δ from g (*parent*$(w),w)$ for every nonroot ancestor w of v.

link(v,w,α,β): Combine the trees containing vertices v and w by making w the parent of v.

Define $g(v,w) = \alpha$ and $g(w,v) = \beta$. Vertices v and w must be in different trees, and v must be a tree root.

cut(v): Break the tree containing vertex v into two by deleting the edge joining v and its parent. Vertex v must not be a root.

evert(v): Reroot the tree containing vertex v by making v the root.

We represent the spanning tree T as a dynamic tree. A tree edge $\{v,w\}$ has $g(v,w) = u_f(v,w)$ and $g(w,v) = u_f(w,v)$. (Note that we can compute one of these values from the other, knowing $u(v,w)$ and $u(w,v)$.) Initialization of T as a dynamic tree requires n *make-tree* operations and $n-1$ *link* operations. This initialization need be done only once, at the beginning of the relaxed network simplex algorithm.

We perform Steps 1a-1c using the dynamic tree operations, as follows. Step 1a requires n *change-mark* operations. There is no need to reroot T; we merely select as r the current root of T.

Step 1b requires the ability to select an admissible arc (r,v) with v unmarked, if any. Since during Steps 1b and 1c no inadmissible arc becomes admissible and no marked vertex becomes unmarked, we can perform this selection by maintaining, for each vertex x, a pointer into a list of incident arcs (x,y). To find (r,v), we move the pointer for r along its list of incident arcs, until finding an admissible arc or reaching the end of the list. The total time for all such scans during all iterations of Step 1b is $O(m)$.

If we do not find an admissible arc (r,v), we complete Step 1b by performing a *change-mark* operation. If we find an admissible arc (r,v) that is a tree arc, we complete Step 1b by by performing *evert*(v). The third and most complicated case occurs if we find an admissible non-tree arc (r,v). Then we must pivot on (r,v). We compute $x =$ *find-min*(v) and $\delta = \min\{$*find-value*$(x), u_f(r,v)\}$. We increase the flow on $\Gamma(r,v)$ by performing *change-value*$(v,-\delta)$ and subtracting δ from $u_f(r,v)$. If $u_f(r,v)$ is now zero, we take (r,v) to be the leaving arc of the pivot; Step 1b is complete. If $u_f(r,v)$ is not zero, we take $(x,$ *parent* $(x))$ to be the leaving arc. We modify T appropriately by performing *cut*(x) to delete $\{x,$ *parent*$(x)\}$, followed by *link* $(r,v, u_f(r,v), u_f(v,r))$ to add $\{r,v\}$. This *cut* and *link* have the side effect of rerooting the tree at x, completing Step 1b.

We perform Step 1c by scanning a list of all the vertices, looking for an unmarked one. The total time for all such scans during all iterations of Step 1c is $O(n)$. If we find an unmarked vertex v, we let $x =$ *find-vertex*(v), and we complete Step 1c by performing *evert*(x).

This method requires $O(1)$ dynamic tree operations for each iteration of Step 1b or 1c, and the total time for Step 1 is $O(m \log n)$ if the amortized time per dynamic tree operation is $O(\log n)$. The flows on the tree arcs can be recovered at the end of the minimum-cost circulation computation by performing $n-1$ *find-value* and $n-1$ *find-parent* operations.

We shall sketch an implementation of dynamic trees such that each operation indeed takes $O(\log n)$ amortized time. The implementation is that of Sleator and Tarjan [37], extended to allow two real values per edge and a mark bit per node instead of one real value as in [37], and extended to use reversal bits to handle *evert* operations as in [36]. We assume some familiarity with [37]; we shall merely outline the data structure, highlighting the changes needed for our application.

We represent a dynamic tree D by a rooted *virtual tree* V, which contains the same vertices as D but has different structure. Each vertex of V has a *left child* and a *right child*, either or both of which can b missing, and zero or more *middle children*. We call an edge of V *solid* if it joins a left or right child to its parent and *dashed* if it joins a middle child to its parent. Thus V consists of a hierarchy of binary trees, its *solid subtrees*, interconnected by dashed edges. The relationship between D and V is that the parent in D of a vertex x is the symmetric-order successor of x in the solid subtree containing x in V, unless x is last in its solid subtree, in which case its parent in D is the parent in V of the root of its solid subtree. (See Figure 1.) That is, each solid subtree in V corresponds to a path in D from some vertex upward toward the root, with symmetric order in the solid subtree corresponding to the order along the path from deepest to shallowest vertex. We say that a vertex x is a *solid descendant* of a vertex y in V, and y is a *solid ancestor* of x, if x is a descendant of y and the path from x to y consists of solid edges.

[Figure 1]

We represent the structure of V by storing with each vertex x pointers to its parent in V, denoted by V-*parent*(x), and pointers to its left and right children, denoted by *left*(x) and *right*(x). These pointers allow us not only to move up and down through V but also to test in $O(1)$ time whether a vertex x is the root of a solid subtree; such is the case if and only if *left*$(V$-*parent*$(x)) \neq x$

and *right*(*V*-parent(x)) $\neq x$. We store two bits with each vertex to encode marking information. For a node x, bit *mark*(x) is 1 if x is marked; bit *all-mark*(x) is 1 if all solid descendants of x are marked.

We store additional information with vertices to encode edge values. This information differs slightly from that used in [37] because the values are associated with edges of D rather than vertices. We associate an ordered pair $(s(x), t(x))$ with each nonroot vertex x of V, as follows: if x is a left or middle child, $t(x)$ is *V-parent*(x) and $s(x)$ is the last solid descendant of x in symmetric order; if x is a right child, $s(x)$ is *V-parent*(x) and $t(x)$ is the first solid descendant of x in symmetric order. The correspondence $x \longleftrightarrow \{s(x), t(x)\}$ is 1-1 between the nonroot vertices of V and the edges of D. We associate with each nonroot vertex x of V four values:

$$g(x) = g(s(x), t(x));$$

$$mg(x) = \min\ \{g(y) \mid y \text{ is a solid descendant of } x\};$$

$$h(x) = g(t(x), s(x));$$

$$mh(x) = \min\ \{h(y) \mid y \text{ is a solid descendant of } x\}$$

We store these values in difference form. Specifically, we store the following four values with each nonroot vertex x:

$$\Delta g(x) = \begin{cases} g(x) \text{ if } x \text{ is a middle child;} \\ g(x) - g(V\text{-}parent(x)) \text{ otherwise;} \end{cases}$$

$$\Delta mg(x) = \Delta g(x) - mg(x);$$

$$\Delta h(x) = \begin{cases} h(x) \text{ if } x \text{ is a middle child;} \\ h(x) - h(V\text{-}parent(x)) \text{ otherwise;} \end{cases}$$

$$\Delta mh(x) = \Delta h(x) - mh(x).$$

We call the values $g(x)$, $\Delta g(x)$, $mg(x)$, $\Delta\, mg(x)$ the *g-values* of x and define the term *h-values* analogously.

Finally, we store with each vertex x a reversal bit $rev(x)$, whose interpretation is a follows. Let $srev(x)$ be the mod-two sum of the reversal bits of all solid ancestors of x. If $srev(x)$ is 1, the true meaning of the left and right child pointers of x is reversed, as is the meaning of the fields holding the g-values and h-values. That is, the left pointer points to the right child and vice-versa; the g-fields hold the h-values and vice-versa.

We perform the dynamic tree operations with the aid of two $O(1)$-time restructuring primitives on virtual trees. The first is *rotation*, in which two vertices x and y joined by a solid edge are interchanged while preserving symmetric order. If x is the left (right) child of y before the rotation, y becomes the right (left) child of x after the rotation, and the right (left) child of x becomes the left (right) child of y; all other vertices retain the same parents. (See Figure 2.) It is straightforward to verify that all the fields of all the vertices can be updated in $O(1)$ time; the only subtle point is that the ordered pairs associated with some of the nodes change. Specifically, if before the rotation x is the left child of y, then the pair originally associated with x becomes associated with the original right child of x, the pair originally associated with the original right child of x becomes associated with y, and the pair originally associated with y becomes associated with x. Nonetheless, all the required updates of values are in vertices within two steps of x, as are all the values needed for the updates.

[Figure 2]

The second restructuring primitive is *splicing*, in which a vertex y that is the root of a solid subtree has its left child, if any, changed to a middle child, and possibly has a middle child changed to its left child. (See Figure 3.) A splice requires less work than a rotation, primarily because none of the ordered pairs associated with the vertices change. This primitive also has an $O(1)$ time bound.

[Figure 3]

Out of rotations and splices we build the main restructuring operation on a virtual tree V, called *splaying*. A splay at a vertex x consists of a specific sequence of rotations and splices done along the path from x to the root of V. The effect of the splaying is to restructure V, making x the root. The time required for splaying is proportional to the original depth of x; the amortized time is $O(\log n)$, if n is the number of vertices in V. The paper [37] gives the details of how splaying is performed and the analysis leading to the $O(\log n)$ amortized time bound.

We can perform each of the dynamic tree operations using at most two splayings and $O(1)$ additional tree restructuring. We shall describe the implementation of two of the operations; the implementation of the others is similar. To perform *evert*(v), we splay at v, perform a splice to make the left child of v (if any) a middle child, and flip the bit *rev*(v). We perform ***find-min***(v) as follows. First, we splay at v. Now the path from v to the root in its dynamic tree D is represented by the right subtree of v in its virtual tree V. If x is the right child of v, the value $\alpha = mg(x)$ is the minimum of $g(w, parent(w))$ for w an ancestor of v in D. We walk down from x through solid descendants, using g-values to guide the search, until reaching a vertex y such that $g(y) = \alpha$. If y is a right child, our search is complete; the vertex to be returned is $w = V\text{-}parent(y)$. If, on the other hand, y is a left child, we continue down the tree through right children until reaching a vertex w with no right children. In either case we complete the ***find-min*** operation by splaying at w and returning w. (The splay pays for the search to reach w.)

The dynamic tree data structure gives us an $O(m\log n)$-time implementation of Step 1 of the relaxed network simplex algorithm, by Lemma 5.5. This result, combined with Theorem 5.3, gives us our final result:

Theorem 5.6. The relaxed network simplex algorithm, implemented with dynamic trees, runs in $O(nm\log n \cdot \min\{\log(nC), m\log n\})$ time.

6. Remarks

There are two major open questions related to our work, one theoretical, one practical. On the theoretical side, we have not resolved the question of whether the network simplex algorithm (with only cost-decreasing pivots) has a version requiring only polynomially many pivots. Our subexponential upper bound suggests the possibility of a nonpolynomial but subexponential lower bound. We are not willing to offer a conjecture as to whether the actual bound is polynomial or not, but we believe that the most fruitful way to approach the problem is to spend as much time in trying to construct a nonpolynomial class of examples as in trying to obtain a polynomial bound. Whatever the answer, the results of Section 4 show that the key to solving the problem lies in obtaining a tight bound on the number of pivots needed for cycle blocking.

On the practical side, one may ask whether any of our results or ideas can contribute to obtaining faster computer programs implementing the network simplex algorithm or other algorithms. The current computational wisdom is that the network simplex algorithm performs very well in practice even if no special effort is made to choose pivots wisely [23]. When advances in computing technology make it possible to attack problems on very large networks, however, some of our ideas may become useful.

In particular, the dynamic tree data structure can be extended so that it can be used to represent the spanning tree in the standard implementation of the network simplex algorithm. It is necessarily merely to add values to the vertices to represent arc costs, and to add an additional operation that sums arc costs along a tree path. Then the amortized time to compute the reduced cost of an arc or to perform a pivot is $O(\log n)$. Standard implementations take $O(n)$ time per pivot but $O(1)$ time to compute the reduced cost of an arc. With storage of one additional value per dynamic tree vertex, the time to compute the reduced cost of k arcs, with no intervening pivots, can be reduced to $O(\min\{k\log n, k+n\})$. We leave filling in the details of this result as an exercise.

7. Acknowledgements

My warmest thanks to Mike Grigoriadis, Éva Tardos, and especially Andy Goldberg, for extensive discussions that contributed greatly to this paper.

8. References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, MA, 1974.

[2] R. K. Ahuja, A. V. Goldberg, J. B. Orlin, and R. E. Tarjan, "Finding minimum-cost flows by double scaling," *Math. Prog.*, submitted.

[3] A. Ali, R. Helgason, J. Kennington, and H. Lall, "Primal network simplex codes: state of the art implementation technology," *Networks* 8 (1978), 315-359.

[4] D. P. Bertsekas, "A distributed algorithm for the assignment problem," unpublished working paper, Laboratory for Information and Decision Sciences, Massachusetts Institute of Technology, Cambridge, MA, 1979.

[5] D. P. Bertsekas, "Distributed asynchronous relaxation methods for linear network flow problems," Technical Report LIDS-P-1986, Laboratory for Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1986.

[6] R. G. Bland and D. L. Jensen, "On the computational behavior of a polynomial-time network flow algorithm," Technical Report 661, School of Operations Research and Industrial Engineering, Cornell University, 1985.

[7] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North Holland, New York, NY, 1976.

[8] K. S. Booth and G. S. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms," *J. Computer and System Sciences* 13 (1976), 335-379.

[9] K. H. Borgwardt, "The average number of pivot steps required by the simplex method is polynomial," *Zeitshrift für Operations Research* 26 (1982), 157-177.

[10] R. G. Busacker and P. J. Gowen, "A procedure for determining a family of minimal-cost network flow patterns," O.R.O. Technical Paper 15, 1961.

[11] R. G. Busacker and T. L. Saaty, *Finite Graphs and Networks: An Introduction with Applications*, McGraw-Hill, New York, NY, 1965.

[12] A. Charnes and W. W. Cooper, "The stepping stone method of explaining linear programming calculations in transportation problems," *Management Science* 1 (1954), 49-69.

[13] G. Dantzig, "Application of the simplex method to a transportation problem," *Activity Analysis of Production and Allocation*, T. C. Koopmans, ed., John Wiley, New York, NY, 1951.

[14] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. Assoc. Comput. Mach.* 19 (1972), 248-264.

[15] L. R. Ford, Jr., and D. R. Fulkerson, *Flows in Networks*, Princeton University Press,

Princeton, NJ, 1982.

[16] S. Fujishige, "A capacity-rounding algorithm for the minimum-cost circulation problem: a dual framework of the Tardos algorithm," *Math. Prog.* 35 (1986), 298-308.

[17] D. R. Fulkerson, "An out-of-kilter method for minimal cost flow problems," *J. SIAM* 9 (1961), 18-27.

[18] Z. Galil and É. Tardos, "An $O(n^2 \log n(m + n\log n))$ minimum cost flow algorithm, *J. Assoc. Comput. Mach.* 35 (1988), 374-386.

[19] A. V. Goldberg, M. D. Grigoriadis, and R. E. Tarjan, "Efficiency of the Network Simplex Algorithm for the Maximum Flow Problem," to appear.

[20] A. V. Goldberg and R. E. Tarjan, "Finding minimum-cost circulations by successive approximation," *Math. of Oper. Res.*, to appear.

[21] A. V. Goldberg and R. E. Tarjan, "Finding minimum-cost circulations by canceling negative cycles," *J. Assoc. Comput. Mach.*, submitted.

[22] D. Goldfarb and J. Hao, "A primal network simplex algorithm that solves the maximum flow problem in at most nm pivots and $O(n^2m)$ time," manuscript, Department of Industrial Engineering and Operations Research, Columbia University, New York, NY, 1988.

[23] M. D. Grigoriadis, "An efficient implementation of the network simplex method," *Math. Prog. Study* 26 (1986), 83-111.

[24] J. E. Hopcroft and R. E. Tarjan, "Efficient planarity testing," *J. Assoc. Comput. Mach.* 21 (1974), 549-568.

[25] M. Iri, "A new method of solving transportation-network problems," *J. Op. Res. Soc. Japan* 3 (1960), 27-87.

[26] W. S. Jewell, "Optimal flow through networks," Interim Technical Report 8, Massachusetts Institute of Technology, Cambridge, MA, 1958.

[27] R. M. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete Math.* 23 (1978), 309-311.

[28] R. M. Karp and J. B. Orlin, "Parametric shortest path algorithms with an application to cyclic staffing," *Discrete Applied Math.* 3 (1981), 37-45.

[29] M. Klein, "A primal method for minimal-cost flows with applications to the assignment and transportation problems," *Management Science* 14 (1967), 205-220.

[30] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Reinhart, and Winston, New York, NY, 1976.

[31] A. Orden, "The transshipment problem," *Management Science* 2 (1956), 276-285.

[32] J. B. Orlin, "Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem," Sloan Working Paper 1615-84, Massachusetts Institute of Technology, Cambridge, MA, 1984.

[33] J. B. Orlin, "A faster strongly polynomial minimum cost flow algorithm," *Proc Twen-*

tieth Annual ACM Symp. on Theory of Computing (1988), 377-387.

[34] J. B. Orlin and R. K. Ahuja, "New scaling algorithms for the assignment and minimum cycle mean problems," Sloan Working Paper 2019-88, Massachusetts Institute of Technology, Cambridge, MA, 1988.

[35] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[36] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *J. Computer and System Sciences* 26 (1983), 362-391.

[37] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *J. Assoc. Comput. Mach.* 32 (1985), 652-686.

[38] S. Smale, "On the average number of steps of the simplex method of linear programming," *Math. Prog.* 27 (1983), 241-262.

[39] É. Tardos, "A strongly polynomial minimum cost circulation algorithm," *Combinatorica* 5 (1985), 247-255.

[40] R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[41] R. E. Tarjan, "Amortized computational complexity," *SIAM J. Alg. Disc. Meth.* 6 (1985), 306-318.

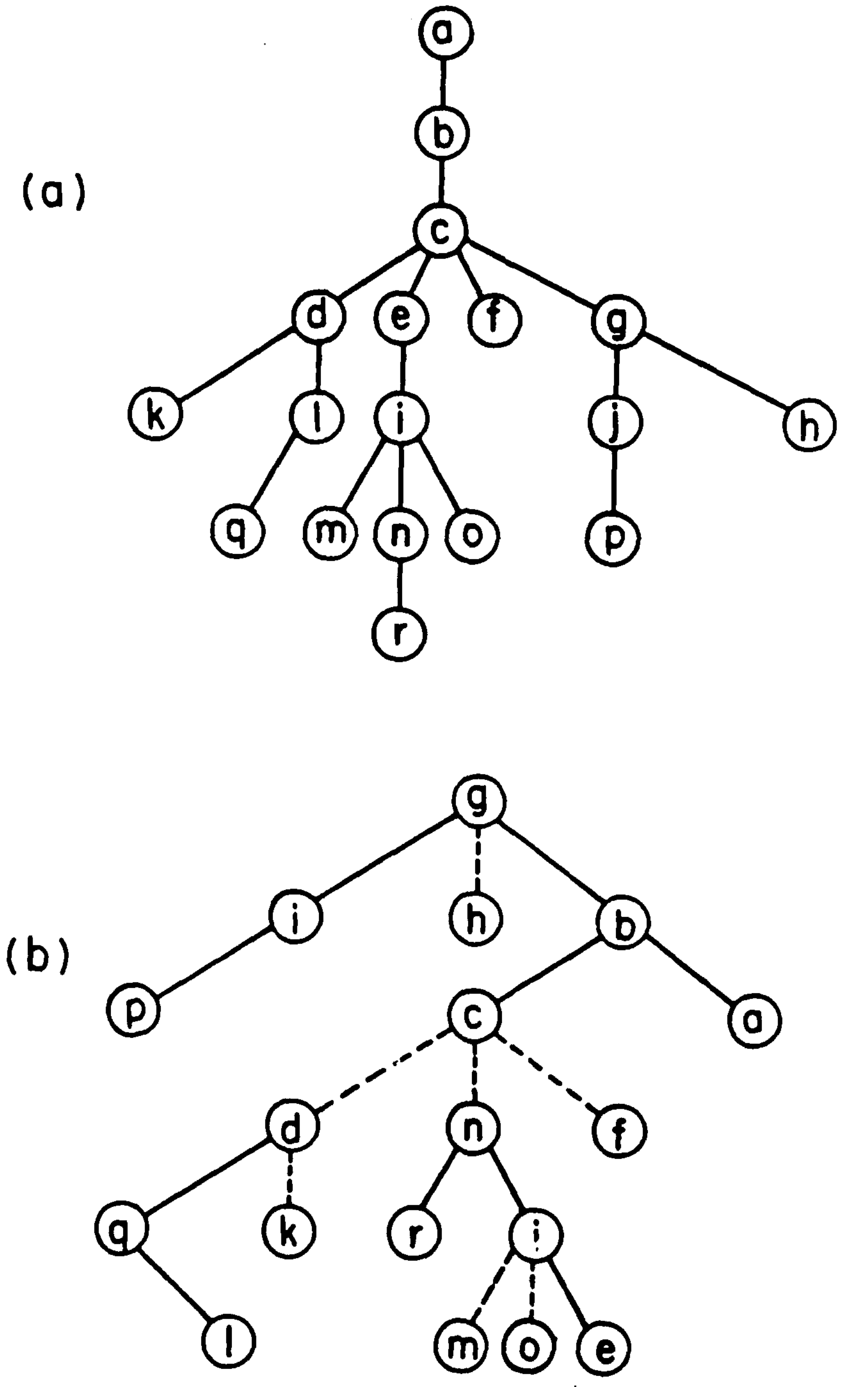

Figure 1. A dynamic tree and the correpsonding virtual tree.
(a) Dynamic tree T (b) Virtual tree V. Solid edges are solid; dashed edges are dashed.

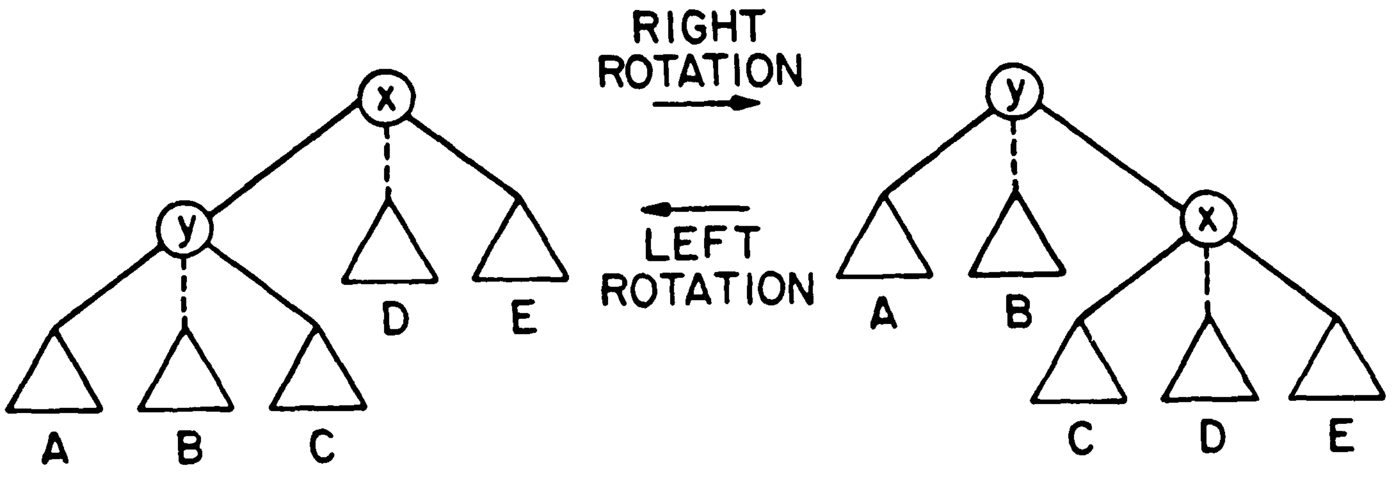

Figure 2. A rotation in a virtual tree. Triangles denote subtrees.

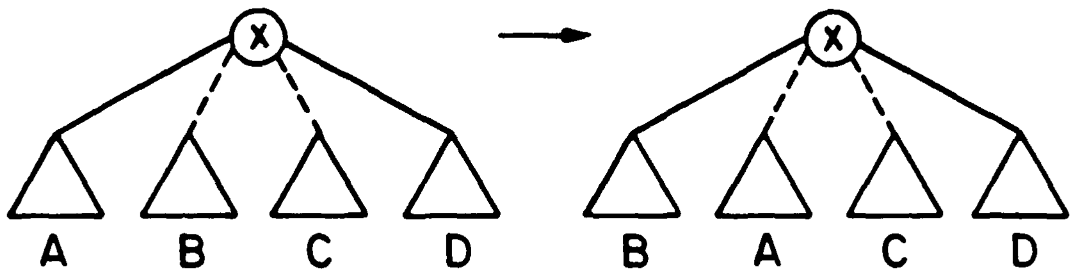

Figure 3. A splice in virtual tree.